Exploring Traditional and Emerging Parallel Programming Models using a Proxy Application

Ian Karlin, Abhinav Bhatele, Jeff Keasler, Bradford L. Chamberlain, Jonathan Cohen, Zachary DeVito, Riyaz Haque, Dan Laney, Edward Luke, Felix Wang, David Richards, Martin Schulz, Charles H. Still

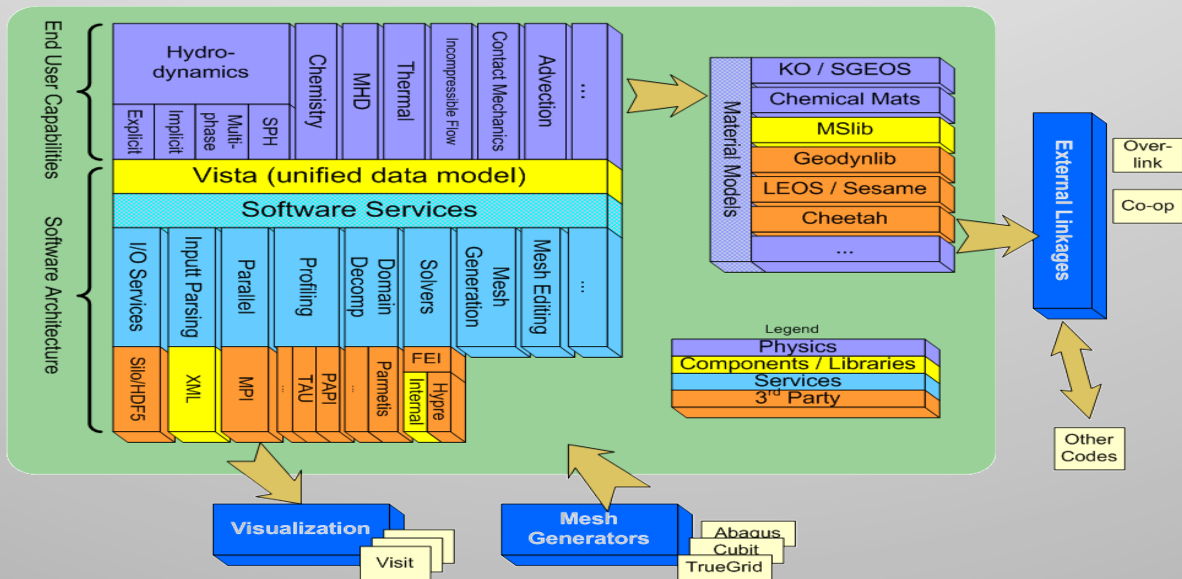
IPDPS ’13 / May 23, 2013

Lawrence Livermore National Laboratory
MISSISSIPPI STATE UNIVERSITY
CRAY
UCLA

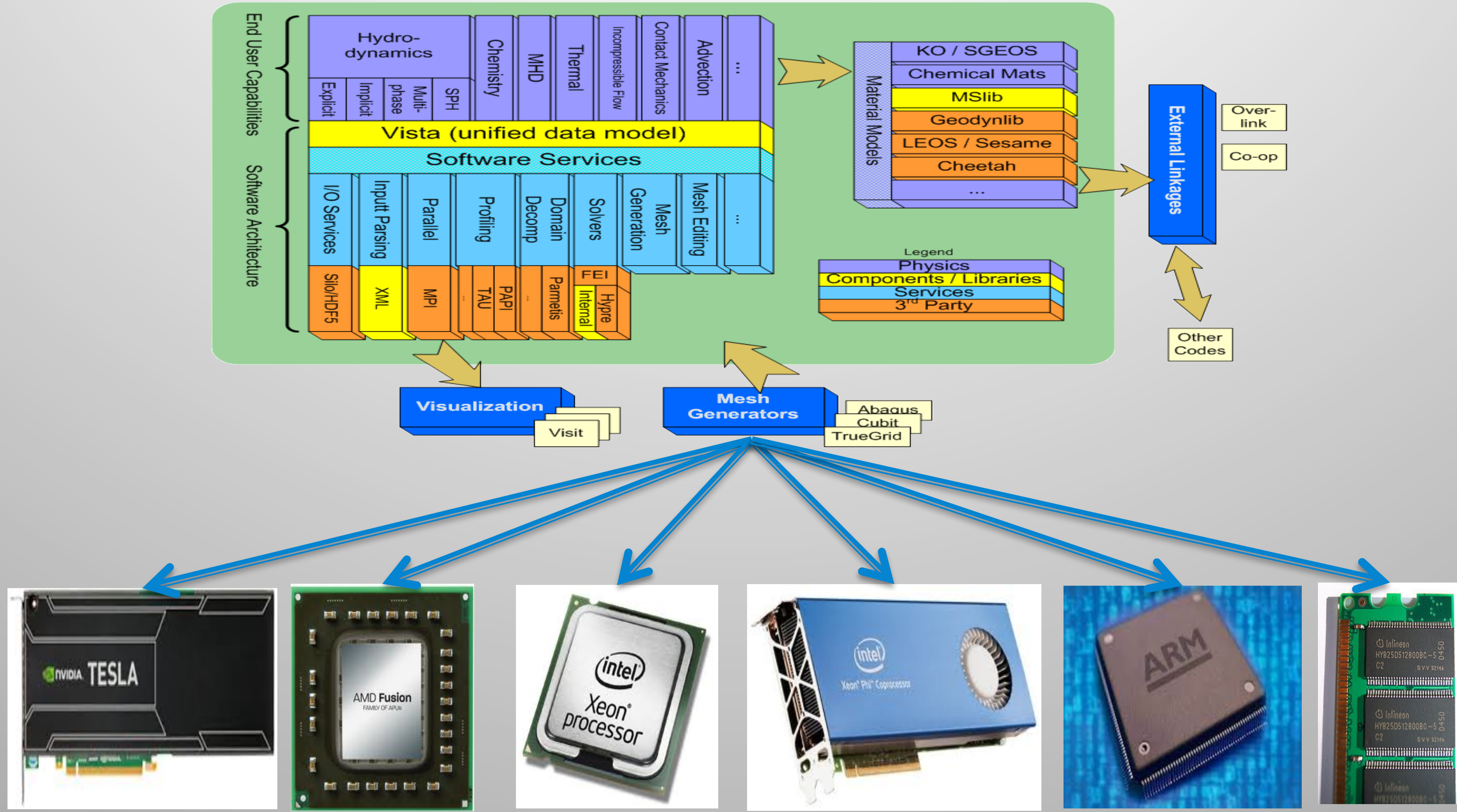
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Motivating Problem

- Currently we cannot afford to tune large complex applications for each hardware
  - Performance
  - Productivity
  - Codebase size



How to Retarget Large Applications in a Manageable Way?



Can New Programming Models Help?

chapel

OpenCL

MPI

NVIDIA CUDA

OpenMP

BERKELEY UPC UNIFIED PARALLEL C

X10

Loci

CHARM++

Liszt

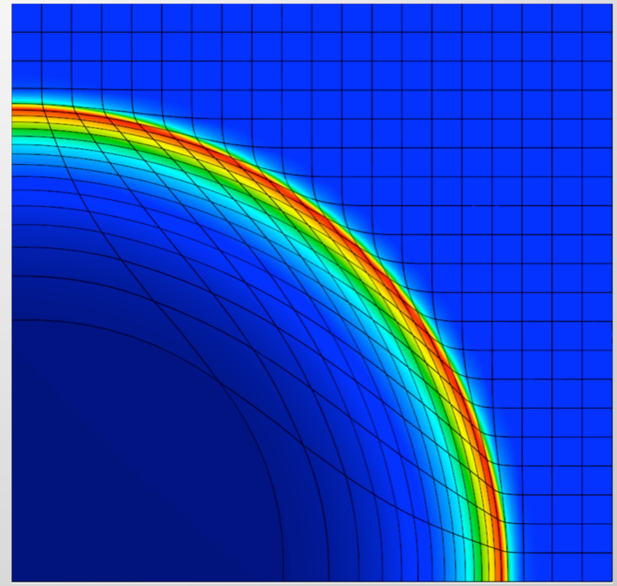
The Questions We Want To Answer

- How can new languages help application portability and maintainability?
- Can applications written in them perform well?
- What is the performance penalty for using them?
- What is needed to get them production ready?

Investigating the use of proxy applications

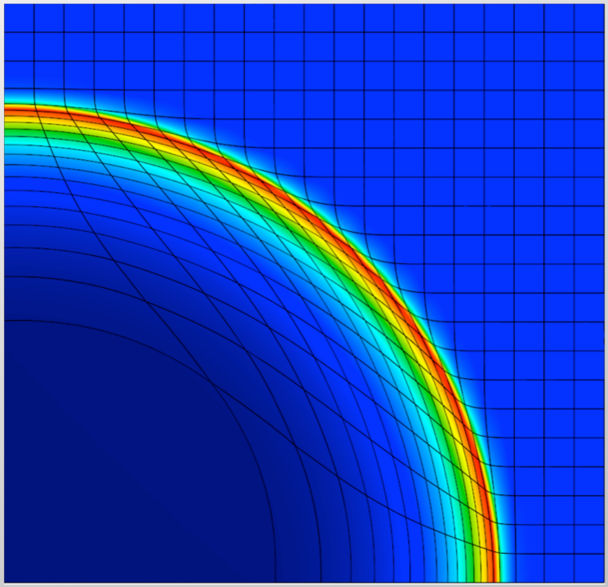
LULESH

- Shock-hydro mini-app
  - Lagrange hydrodynamics
  - Solves Sedov Problem
  - Unstructured hex mesh
  - Single material
  - Ideal gas EOS



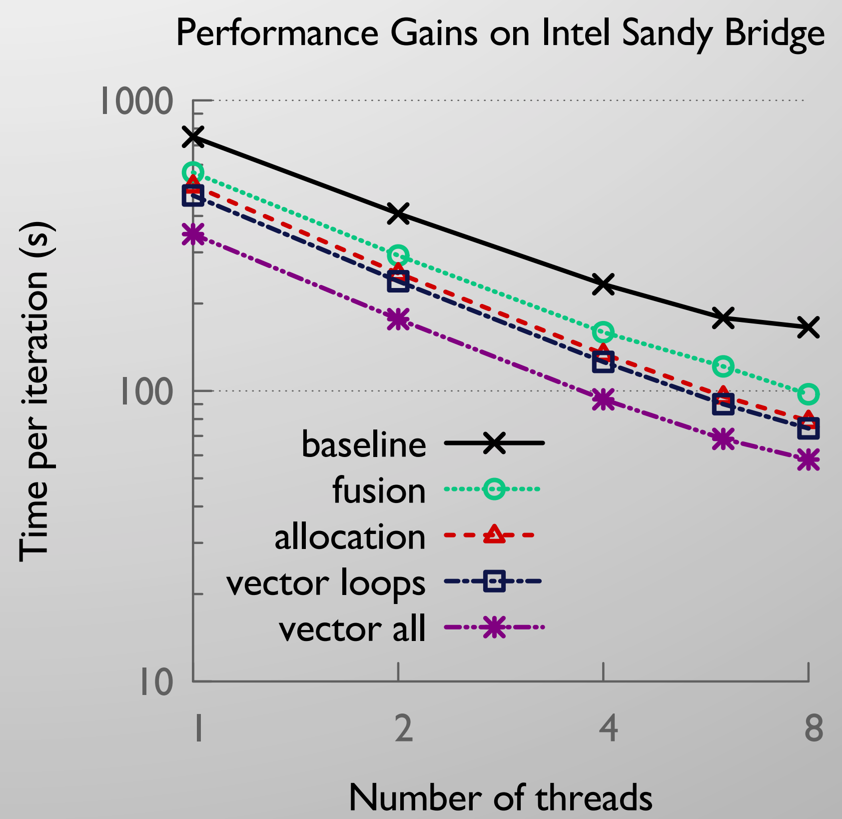
Initial Implementations

- Serial
- OpenMP
- MPI
- Hybrid MPI/OpenMP
- CUDA (Fermi)

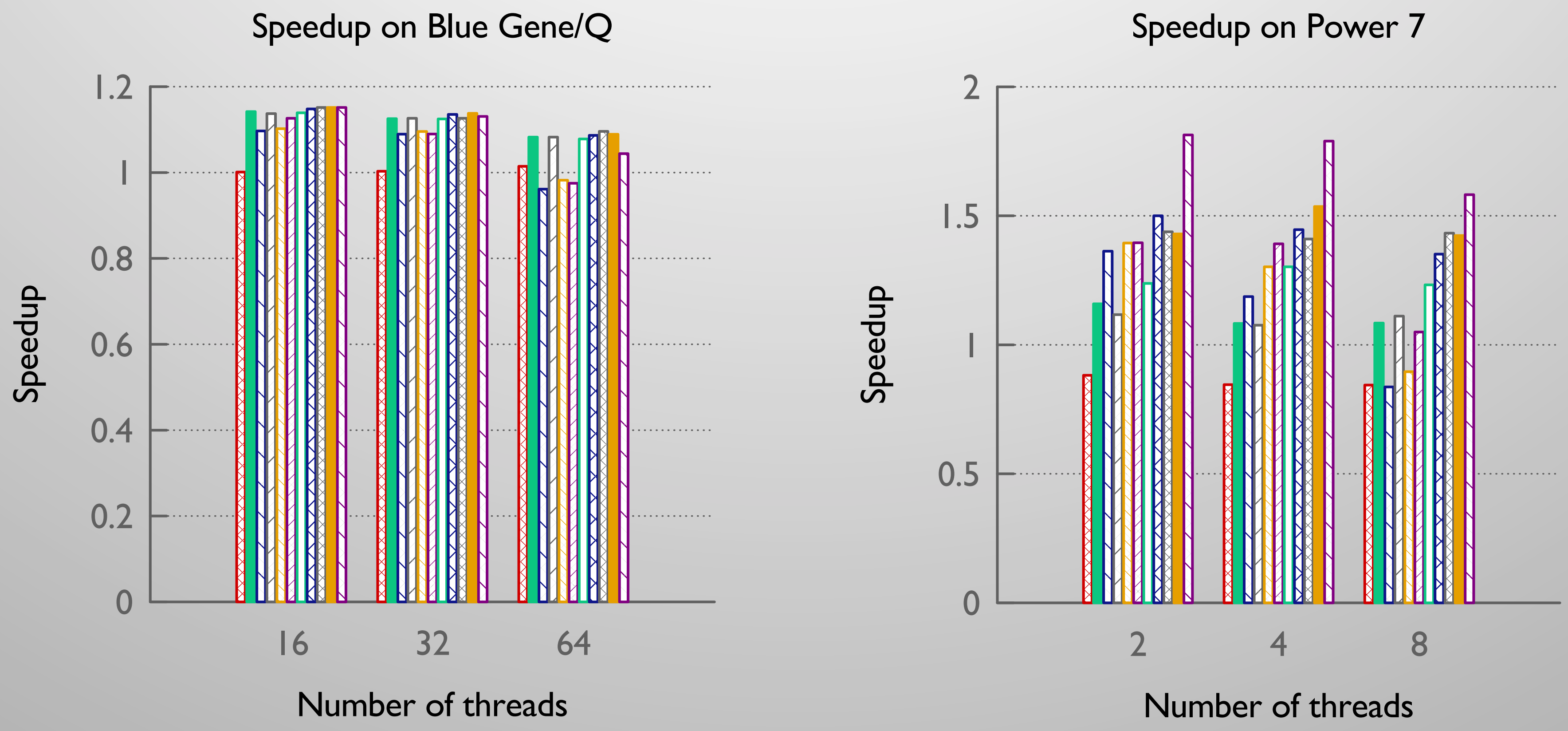


Four Changes Lead to Good On-node Performance Gains

- Loop fusion
- Data structure transformations
- Memory allocation
- Vectorization



The Best Data Layout Depends on the Architecture

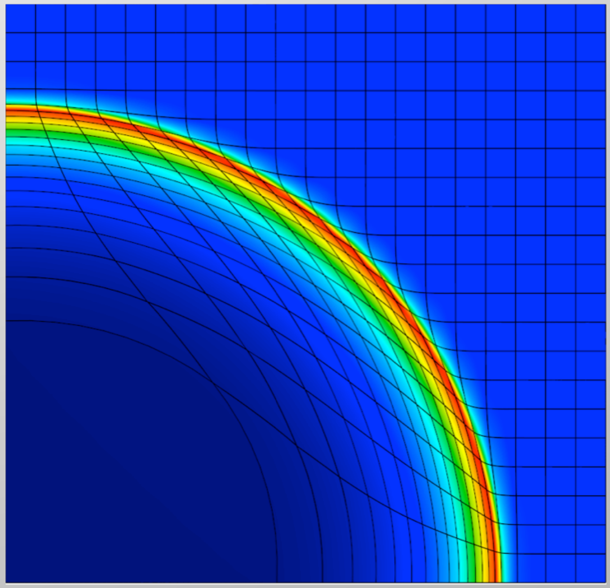


Why This is not Maintainable?

- Porting to various architectures requires refactoring significant amounts of code
- Tuning requires even more extensive changes
- Expert knowledge needed for each architecture
- Maintaining multiple versions of code can lead to bug control and versioning issues

LULESH Programming Model Ports

- Chapel
  - Partitioned global address space (PGAS)
  - Imperative block structured like C/C++/Fortran
- Charm++
  - Builds on C++
  - Message-driven execution
- Loci
  - Functional/relational
  - Dataflow-driven
- Liszt
  - Domain-specific language for PDEs
  - Targets CPUs and GPUs



New Programming Models Result in Smaller Code Bases

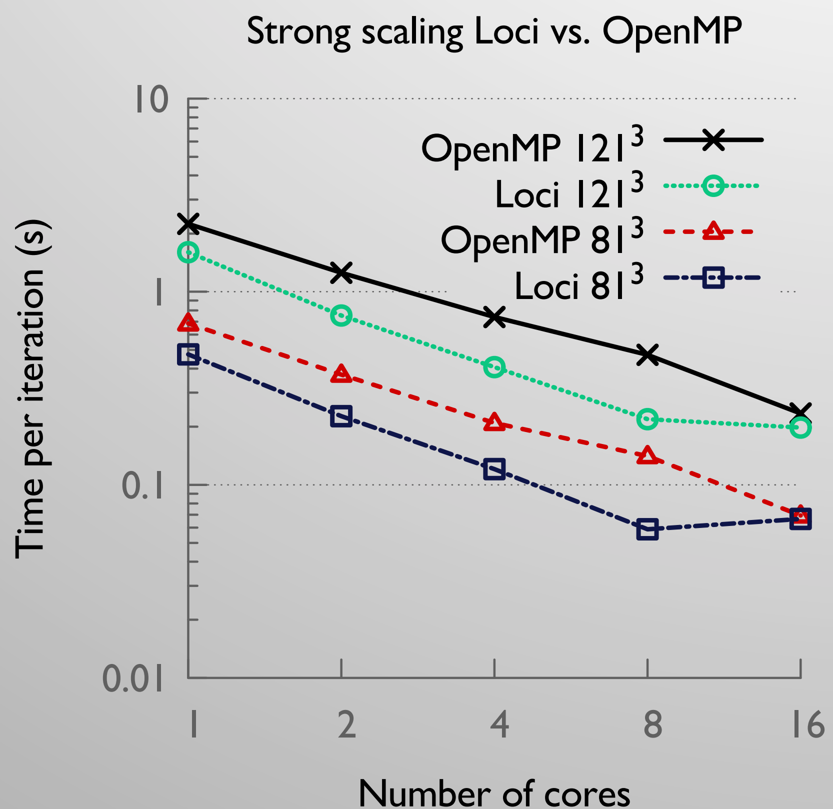
Conventional Models

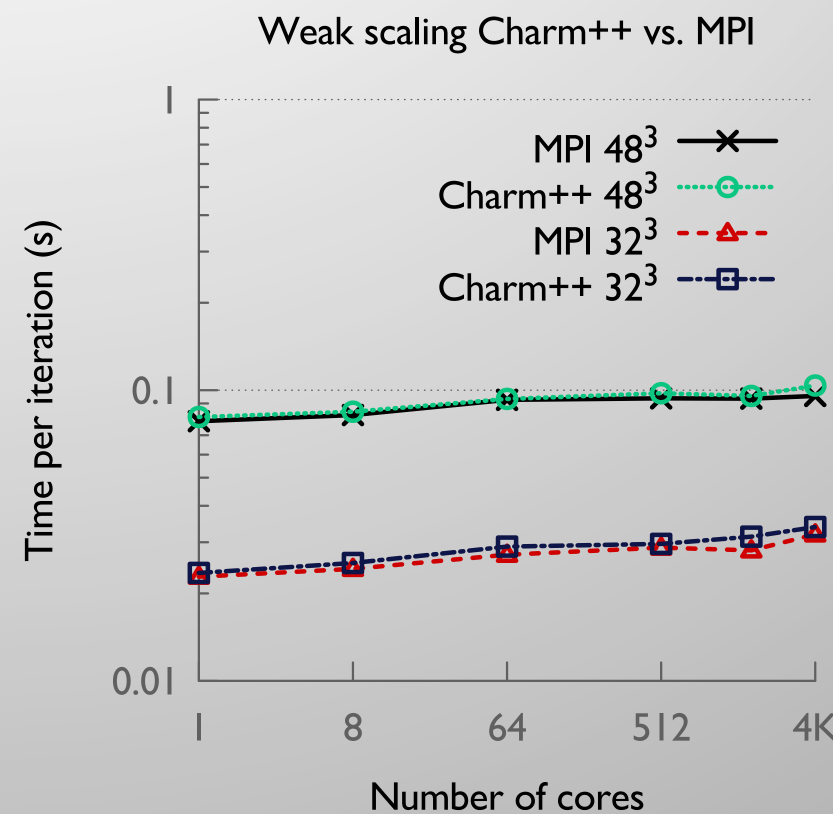
Model	Lines of Code
Serial	2183
OpenMP	2403
MPI	4291
MPI + OpenMP	4476
CUDA	2990

Other Models

Model	Lines of Code
Chapel	1108
Charm++	3922
Liszt	1026
Loci	742

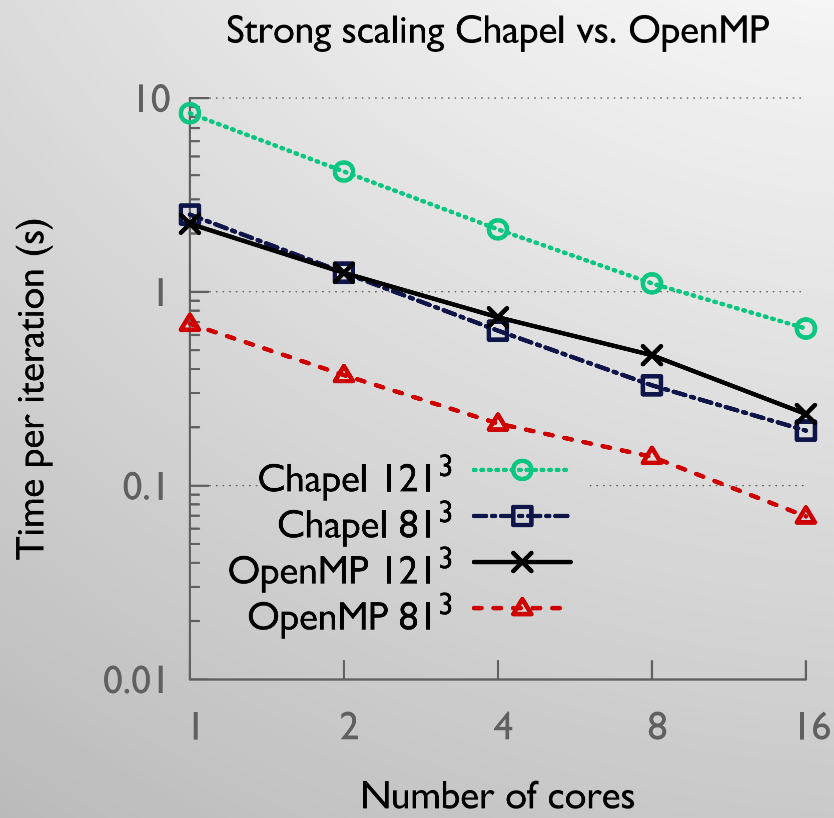
Untuned versions of Loci and Charm++ Produce Good Performance

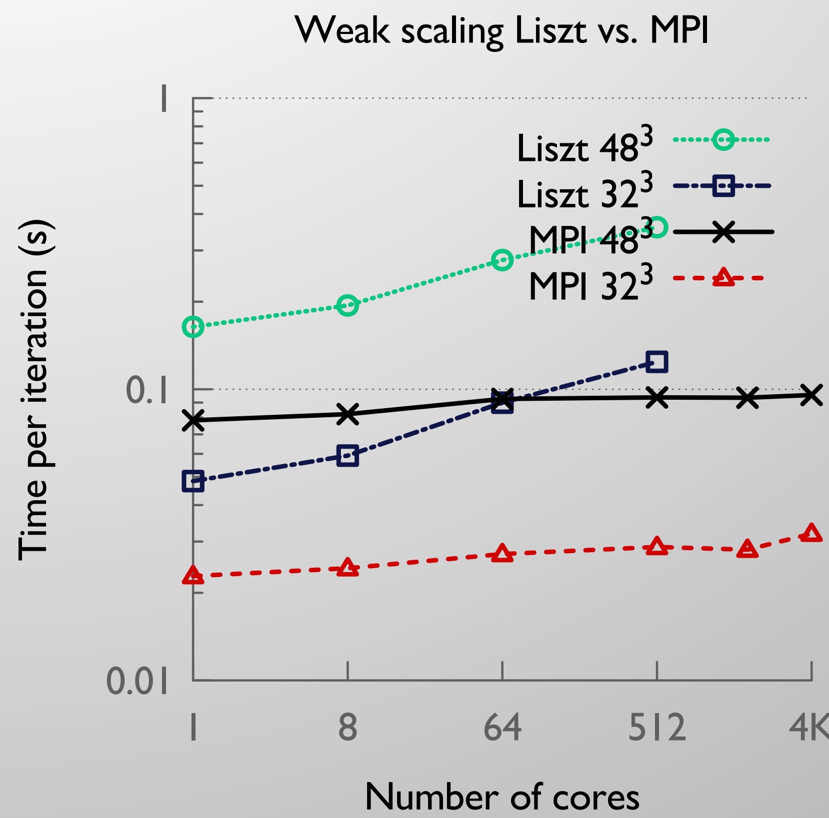




Intel Sandy Bridge cluster at LLNL (Cab)

Other Models Produce Good Scalability





Intel Sandy Bridge cluster at LLNL (Cab)

Performance will improve as models mature

Transformations Applicable to LULESH

Model	Loop Fusion	Data Structure Trans.	Global Allocation	SIMD
Chapel		☑		
CHARM++				
Liszt	☑	☑	☑	*
Loci		☑	☑	*

Other Transformations

Model	Blocking	Overlap
Chapel	☑	☑
CHARM++	☑	☑
Liszt	*	
Loci	☑	☑

Other features, such as, load balancing and fault tolerance available in some languages, but outside this paper's scope.

New Prog. Models Make Data Structure Transformations Less Invasive

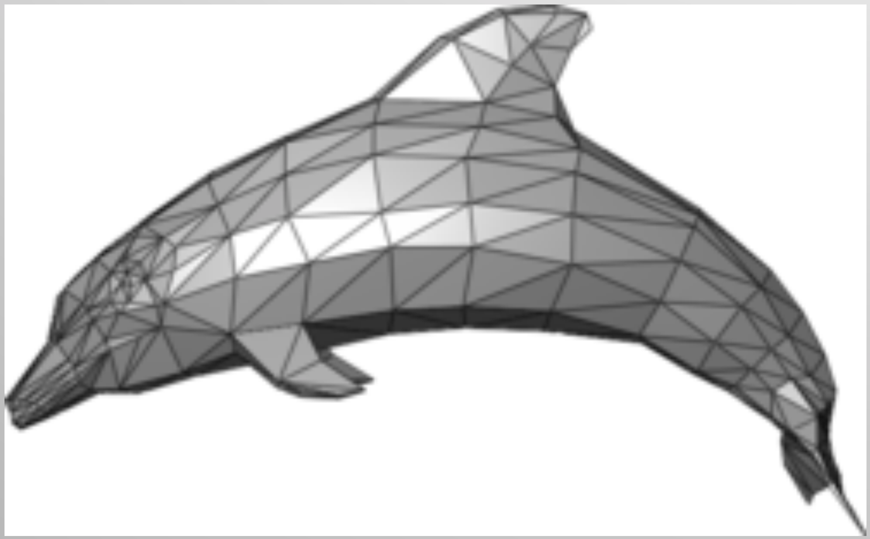
```
Real x[n];
Real y[n];
Real z[n];
```

→

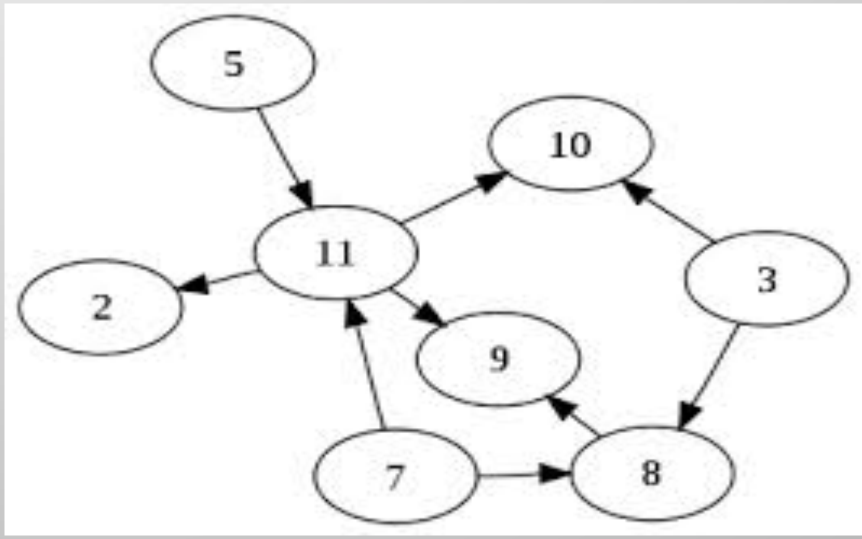
```
Struct xyz {Real x,y,z;}
coords xyz[n];
```

Additional Information Can Help the Compiler Generate SIMD

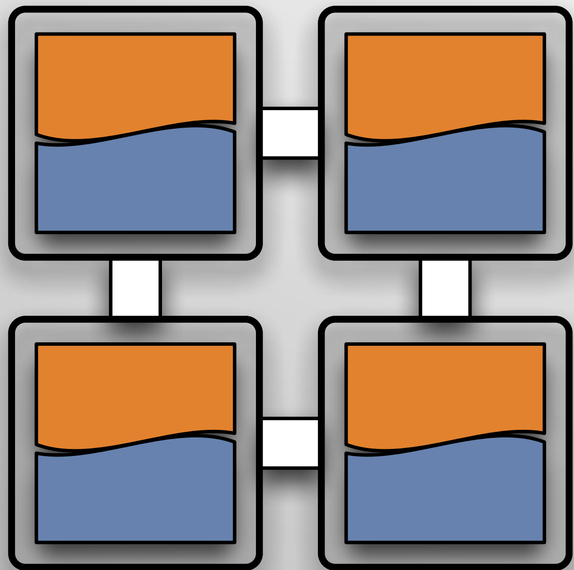
Liszt knows a mesh is being used



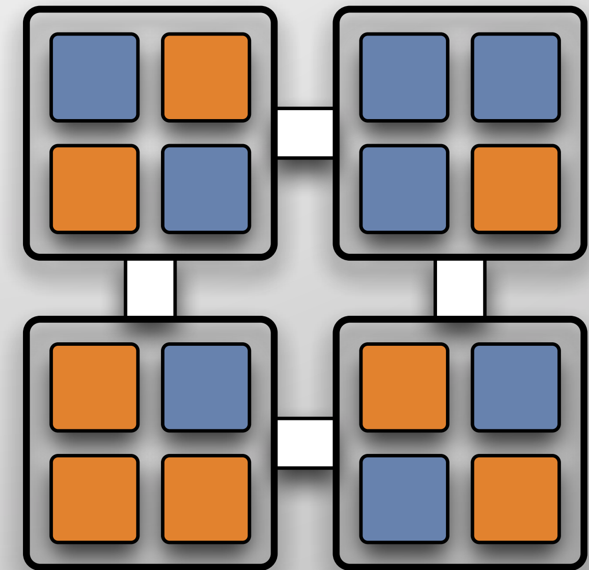
Loci knows more dependence information



Over Decomposition Enables Blocking and Overlap in Charm++



4 MPI processes
on 4 processors



16 Charm++ objects
on 4 processors

There is hope …

- Performance is possible with newer approaches
- New models add features that enable portable performance
- Smaller codebases that are easier to read and possibly maintain
- However, we need more features for general use

Co-Design to Improve Chapel

- Original port by Cray assumed that the mesh is structured
  - Block -> Unstructured change ~ 6 hours
  - 25 extra lines of code!
- Now supports fully unstructured meshes
- LULESH is now part of Chapel test suite.

Co-Design to Improve Liszt

- First compute-intensive code ported
  - Identified areas to improve the language
    - New abstractions
    - Fine-grained control over data and workload distribution
- Work led to the motivation for Tera

Co-Design to Improve Loci

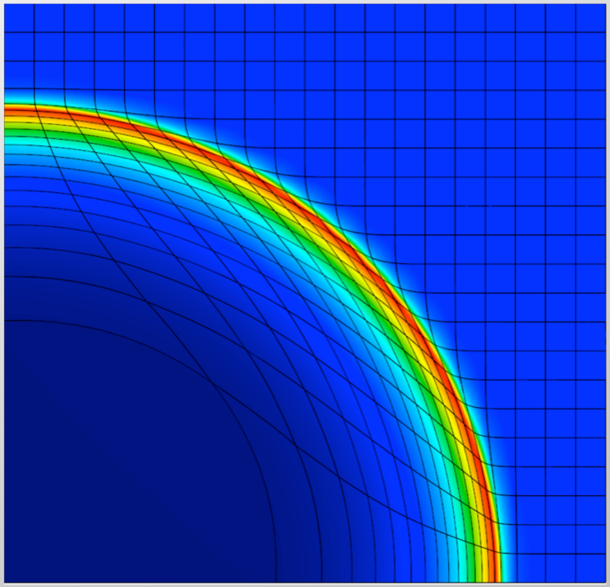
- Implemented additional support for hexahedral zones
- Improvements to message scheduler
- Found two bugs in the underlying communication

Takeaway Lessons

- New models have many attractive features for portable performance.
- Some have performance comparable or better to a C/C++ implementation.
- Application scientist and model developer co-design leads to mutually beneficial improvements.

Continuing Work

- Exploration of other models:
  - OpenACC
  - OpenCL
  - UPC
- LULESH 2.0
  - Multi-region physics
  - Adds load imbalance
  - Charm++ port planned
  - Tera port planned



Takeaway Lessons

- New models have many attractive features for portable performance.
- Some have performance comparable to or better than a C/C++ implementation.
- Co-design by application scientists and language/prog. model developers leads to mutually beneficial improvements.

https://codesign.llnl.gov/lulesh.php